TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B 2612 – Elektrotechnika a informatika
Studijní obor: 2612R011 – Elektronické informační a řídicí systémy

Volně šiřitelné XSLT procesory

Open source XSLT processors

Bakalářská práce

Autor: **Rostislav Rež**
Vedoucí práce: RNDr. Pavel Satrapa, Ph.D.

V Liberci 1.1. 2007

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Katedra aplikované informatiky Akademický rok: 2006/2007

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jméno a příjmení: Rostislav Rež

studijní program: B 2612 – Elektrotechnika a informatika

obor: 2612R011 - Elektronické informační a řídicí systémy

Vedoucí katedry Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto bakalářskou práci:

Název tématu: **Volně šiřitelné XSLT procesory**

Zásady pro vypracování:

1. Stručně popište základy jazyka XSLT a jeho zpracování.
2. Vyhledejte volně šiřitelné XSLT procesory, charakterizujte jejich vlastnosti, schopnosti a omezení.
3. Vypracujte testovací dokumenty a stylové předpisy a ověřte na nich chování jednotlivých procesorů. Při testování ověřte i podporu národních znaků pro dokumenty v češtině. Otestujte také výkon procesorů.
4. Vyhodnoťte dosažené výsledky a doporučte vhodný procesor pro zpracování českých textů.

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé BP a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum : 1. 1. 2007

Podpis : ……

Poděkování

Touto cestou bych chtěl poděkovat RNDr. Pavlu Satrapovi, Ph.D, vedoucímu mé bakalářské práce za jeho ochotu, spolupráci, odborné rady či připomínky a především za čas, který mi věnoval v průběhu psaní mé bakalářské práce. Dále mé poděkování patří oběma mým rodičům za trpělivou podporu.

Rostislav Rež

Abstrakt

Hlavním cílem této bakalářské práce je vyhledání volně šiřitelných XSLT procesorů, vypracování testovacích dokumentů a stylových předpisů a jejich následné použití při testování jednotlivých procesorů.

V bakalářské práci se budu věnovat jednak teoretické části, tedy základům XML a popisu jazyka XSLT a dále pak popisu jednotlivých XSLT procesorů. Hlavní část je věnována přípravě testovacích dokumentů a samotnému testování.

Testování procesorů jsem rozdělil na porovnání jejich výkonů při transformacích velkého dokumentu, několika středně velkých a většího počtu malých. Dále jsme procesory testoval se složitými docbookovými styly. Detailní pohled je věnován testování podpory národních znaků, v rámci kterého jsem ověřil schopnost procesorů tvořit dokumenty v různých formátech.

Klíčová slova – XSLT procesor, testování, volně šiřitelné, XML, XPath

Abstract

The main target of this work is to find open source XSLT processors, to make up test documents and stylesheets and further to use them for testing of processors.

In my Bachelor's work I applied to theoretic part, which contains basics of XML and short description of XSLT language. Next I described each XSLT processors. The main part of my work is dedicated to preparing of test documents and stylesheets and to testing of processors.

The testing of processors I split to compare their performance in transformations of one big document, a few middle-sized documents and more small documents. Next I tested each processor with complex DocBook stylesheets. Auxiliary view is dedicated to testing of support of national charsets. Within the frame of this testing is checking capability of processors to create documents in several formats.

Keywords – XSLT processor, testing, open source, XML, XPath

Obsah

Seznam Ilustrací

Úvod

Internet je v současné době nejrychleji expandující médium, o čemž svědčí i poslední odhadovaný počet internetových stránek, který přesahuje sto milionů. Současně s ním se rozvíjejí i jeho technologie, mezi které patří i XSLT.

Tato specifikace velmi úzce souvisí s formátem XML, který se těší stále větší oblibě u tvůrců článku, odborných prací, či webových prezentací. A to díky jeho otevřenosti, jednoduchosti a nezávislosti na platformě.

Hlavní náplní této bakalářské práce je popis a testování „open source“, neboli volně šiřitelných XSLT procesorů.

První kapitola obsahuje úvod do problematiky XSLT, seznámení s konsorciem W3C a popis historie a vývoje standardu XSLT. Náplní druhé kapitoly je seznámení s jazykem XML, jeho syntaktickými pravidly a způsoby tvorby XML dokumentů. To zahrnuje i stručný popis významu a použití DTD. Třetí kapitola popisuje tvorbu XSLT stylů, seznamuje se stromy a uzly. Dále jsou v ní popsány základní xsl elementy, které se používají při tvorbě šablon. Poslední část třetí kapitoly se věnuje jmenným prostorům. Jazyk XPath, jeho datové typy a výrazy jsou náplní čtvrté kapitoly.

Pátá kapitola se již zabývá vlastními XSLT procesory. Jsou zde konkrétně popsány Xalan, Saxon, xsltproc, XT, Sablotron a Jd.xslt. V následující šesté kapitole je zmíněn hardware a software pro testování, dále jsou zde uvedeny všechny používané verze procesorů, včetně zdrojů, odkud se dají získat a způsobů jejich spouštění a použití.

Sedmá kapitola se zabývá testováním výkonu procesorů, které je rozdělené na testování na velkém souboru, několika středně velkých a na větším počtu malých souborů. Osmá kapitola se věnuje testování se složitými styly, v jejímž úvodu jsou popsány použité soubory, následuje popis testování jednotlivých procesorů. Poslední devátá kapitola, věnující se testování češtiny a výstupních formátů, začíná popisem použitého XML dokumentu a jeho verzí pro různá kódování. Dále jsou zde popsány šablony pro převod do HTML, na textový soubor a šablona pro úpravu XML.

1. Úvod do XSLT

1.1 Co je XSLT?

XSLT (Extensible Stylesheet Language Transformations) je ve své podstatě mechanismus pro transformaci obsahu dokumentů XML (Extensible Markup Language) a manipulaci s ním. Jazyk XML umožňuje strukturovat data uložená v jednotlivých dokumentech, XSLT pak umožňuje s obsahem těchto dokumentů pracovat, manipulovat a na jeho základě vytvářet další dokumenty.

XSLT je součástí širší specifikace, specifikace jazyka XSL (Extensible Stylesheet Language). Jazyk XSL se dělí na dvě částí: formátovací a transformační. Formátovací část, která slouží k určení vizuální podoby, je založena na speciálních formátovacích objektech a je proto označována jako XSL:FO. Transformační část, která se využívá k úpravě a reorganizaci dokumentu, je právě XSLT.

1.2 Konsorcium W3C

Konsorcium W3C (World Wide Web Consorcium) je organizace, která definuje standardy pro síť WWW. Byla vytvořena v roce 1994. Jedním ze zakládajících členů byl Tim Berners Lee, který roku 1989 vynalezl World Wide Web. Bylo to v době, kdy působil v CERN (Conseil Européen pour la Recherche Nucléaire, Evropská organizace pro jaderný výzkum). Členy W3C jsou významné komerční firmy, které činnost této organizace financují.

Konsorcium W3C vydává specifikace pro všechny standardy týkající se WWW a souvisejících technologií, tedy i XML, XSL a XSLT. Tyto specifikace se nenazývají standardy, neboť na základě mezinárodní dohody jsou standardy vytvářeny pouze vládami schválenými organizacemi. W3C tedy vydává požadavky (requirements) na nové specifikace. Tyto požadavky stanoví určité mezníky, náhled na to, jak by měla budoucí specifikace vypadat. Po nějaké době W3C vydává první verzi specifikace jako pracovní koncept (working draft). K takovému konceptu se pak mohou vyjadřovat všichni uživatelé. Po shrnutí všech připomínek se specifikace objevuje jako kandidátní doporučení (candidate recomendation), která také podléhá veřejnému připomínkování. Nakonec se objevuje konečná verze doporučení (recommendation). S doporučeními W3C se zachází de facto jako se standardy pro oblast WWW.

1.3 Historie a vývoj XSLT

Pokud se chceme dostat do historie XSLT, musíme začít u předka dnešního XML – SGML (Standard Generalized Markup Language). Tato specifikace byla vyvinuta již v roce 1980, ale byla tak složitá a nepřehledná, že ji ve skutečnosti používal málokdo. Později byl uveden jazyk DSSSL (Document Style Semantics and Specification Language) sloužící k analýze a zobrazení dokumentů. Ten se měl používat společně s SGML. V roce 1998 byla konsorciem W3C vydána specifikace pro XML, což je vlastně zjednodušená verze SGML. Stejně jako SGML je tedy předek XML, tak i DSSSL je předchůdce dnes používaného XSL. Ke spojení XML a XSL se začalo využívat XSLT, které umožňuje snadnou transformaci dokumentů XML na dokumenty XSL:FO, HTML, RTF apod.

První pracovní koncept pro XSLT byl zveřejněn 18. srpna 1998, konečná verze doporučení XSLT 1.0 byla pak vypracována 13. listopadu 1999. Dodnes je to páteř XSLT. Po této specifikaci byl uvolněn první koncept XSLT 1.1, to bylo 12. prosince 2000. Přestože se měl původně stát novým doporučením, začali někteří členové konsorcia W3C pracovat na verzi 2.0. 24. srpna 2001 se W3C rozhodlo ukončit práce na specifikaci XSLT 1.1. Proto se z něj již nikdy nestane doporučení a nikdy se neobjeví žádná oficiální verze 1.1 jazyka XSLT. Přesto, nebo právě proto, konsorcium W3C prohlašuje, že má v úmyslu integrovat většinu toho, co bylo dosaženo ve verzi 1.1 i do specifikace XSLT 2.0. Specifikace XSLT 2.0, jejíž první pracovní koncept byl vydán již 20. prosince 2001, se v současné době nachází ve fázi „kandidát doporučení“, jejíž poslední verze byla vydána 6. srpna 2006.

1.3.1 Hlavní novinky ve specifikaci XSLT 2.0

- podpora vícenásobných výstupů
- podpora seskupování
- podpora uživatelských funkcí
- možnost využívat proměnné v predikátech
- schopnost číst externí soubory
- převádí tzv. typ result tree na node-set (lze s ním dále pracovat pomocí XPath)

2. Základy jazyka XML

2.1 Úvod do XML

Abychom mohli dále rozebírat jazyk XSLT, jeho transformace a procesory, musíme se seznámit s jazykem XML. Ten tvoří jednu z podstatných částí této problematiky. XML tedy vznikl hlavně na základě neduhů, které se vyskytly u jazyka HTML (HyperText Markup Language). Ten v neoficiální verzi vznikl již v roce 1989, ale oficiálně až v roce 1996, a to rovnou ve verzi 2.0. Poslední verze HTML 4.01 byla vydána 24. prosince1999.

HTML získalo oblibu pro svoji jednoduchost, která byla v ostrém kontrastu se složitostí jazyka SGML, ze které vyšel. Na počátku vývoje měl pouze pár tagů. Jenže čím byl jazyk HTML oblíbenější, tím více se rozšiřovalo množství používaných značek. To plynulo z hlavního problému HTML, kterým je fixní složení značek. HTML totiž nemá obecné značky, ale každý prvek se dá použít pouze pro konkrétní účel.

Dalším problémem je, že k dosažení odpovídající struktury informací je potřeba příliš mnoho tagů (dnes není problém setkat se se stránkou, která obsahuje ve zdrojovém kódu více značek než vlastního textu). Současný vývoj navíc směřuje k tomu, že bude stále více uživatelů přistupovat k webu z jednoduchých prohlížečů implementovaných např. v mobilních telefonech. Ty vyžadují jednodušší značkovací jazyk, který obsahuje jen několik tagů a je jednodušší na zpracování. Další nepříliš dobrou vlastností HTML je to, že je tento jazyk značně benevolentní k chybám ve zdrojovém kódu, což může být výhodné pro tvůrce stránek, ne však pro prohlížeče, které musejí tyto prohřešky řešit.

Všechny tyto problémy řeší právě jazyk XML. Jedná se o meta jazyk, tedy o jazyk určený k definici dalších jazyků. Vytváří nové značky, jejich atributy a pravidla pro jejich používání. Síla XML tedy spočívá především v tom, že nemá žádné předem určené tagy. Tím se řeší minimální počet značek, ale i možnost mít značek libovolné množství. Jazyk XML má také jednoduchá syntaktická pravidla, která se však musejí striktně dodržovat, což umožňuje snazší vývoj prohlížečů, které nemusejí opravovat chyby v syntaxi.

2.2 Syntaktická pravidla jazyka XML

2.2.1 Každý XML dokument je tvořen elementy

XML dokumenty jsou tvořeny elementy, jejichž zápis se skládá z počáteční značky, lomené závorky vlastního obsahu elementu a koncové značky, opět lomené závorky. Koncový tag v sobě má navíc znak lomítka.

<ahoj> obsah elementu </ahoj>

Speciálním případem je element, který nemá žádný obsah:

<ahoj></ahoj>

Ten lze zapsat jen jako: <ahoj/>

2.2.2 Názvy XML elementů odpovídají pravidlům

XML má téměř neomezenou volnost při vytváření jmen elementů. Jediné omezení spočívá v tom, že všechny názvy tagů musí začínat písmenem nebo podtržítkem a dalšími znaky mohou být pouze písmena, číslice, podtržítka, pomlčky a tečky. Jiné znaky nejsou povoleny.

2.2.3 XML dokument obsahuje kořenový element

Všechny elementy XML dokumentu musí být obsaženy v kořenovém elementu, je to element na nejvyšší úrovni a nesmí být obsažen v žádném jiném elementu.

2.2.4 XML rozlišuje malá a velká písmena

To znamená, že například značky <Ahoj>, <AHOJ> a <ahoj> jsou považovány za odlišné.

2.2.5 XML elementy se nesmí křížit

Pokud nějaký element obsahuje počáteční značku jiného elementu, musí obsahovat i jeho příslušnou koncovou značku.

2.2.6 XML atributy odpovídají pravidlům

U každého XML dokumentu se může vyskytovat tzv. atribut, který obsahuje doplňující informace o elementu. Zapisuje se do počáteční značky v následujícím tvaru:

<element atribut=”hodnota atributu”> obsah elementu </element>

Atributy se od názvu elementu oddělují mezerou, proto jména elementů nesmí obsahovat mezeru. Stejně tak se v atributu nesmí vyskytovat uvozovky.

2.2.7 XML dokument nesmí obsahovat zakázané znaky

V XML existují znaky, které mají svůj specifický význam a nelze je tedy do dokumentu libovolně psát. Místo nich lze použít tzv. vestavěné entity. Těmito znaky jsou <,>,&,” a ‘.

Každý XML dokument by měl také obsahovat informaci o verzi, podle které byl dokument vytvořen a kódování. Např.:

<?xml version=”1.0” encoding=”windows-1250”>

2.3 Příklad XML dokumentu

Na následujícím obrázku je příklad XML dokumentu, který obsahuje devět elementů a dva atributy. Kořenovým elementem je prvek *CD*.

```
<?xml version="1.0" encoding="windows-1250"?>
<CD>
        <NAZEV>Empire Strikes Back</NAZEV>
        <INTERPRET>John Williams</INTERPRET>
        <ROK>1980</ROK>
        <CENA MENA="Kc">515</CENA>
        <DELKA JEDNOTKA="minut">45</DELKA>
        <POCETPISNI>12</POCETPISNI>
        <ZANR>Soundtrack</ZANR>
        <LABEL>Varese Sarabande</LABEL>
</CD>
```

Obr. 1 – Příklad XML dokumentu

2.4 Definice Typu Dokumentu XML

Definice Typu Dokumentu (Document Type Definition), neboli DTD, je v podstatě sada pravidel pro tvorbu určitého typu dokumentu. Je to jedna z nejdůležitějších vlastností, které XML podědilo od SGML. Umožňuje nám definovat vlastní sadu značek (elementy a atributy), které budou k dispozici pro XML dokumenty vytvářené podle tohoto DTD. V definici typu dokumentu můžeme určit, kde se dané elementy a atributy mohou vyskytovat a jaký mají obsah či hodnotu.

2.4.1 Deklarace Elementu

Každá deklarace DTD vždy začíná znaky *<!* . Následuje slovo ELEMENT určující, že se jedná o deklaraci elementu. Další položkou je název elementu přesně v tom tvaru, v jakém bude používán v XML dokumentu. Nakonec je v kulatých závorkách zapsáno, co daný element může obsahovat. Deklarace je uzavřená lomenou závorkou

2.4.2 Deklarace Atributu

Jak již bylo uvedeno, každá deklarace začíná znaky *<!* a ani v případě atributu to není jinak Následuje slovo ATTLIST, které určuje, že se jedná právě o deklaraci atributu. Dalším slovem v řádce je jméno elementu, u kterého se daný atribut vyskytuje, po tom je uveden vlastní název atributu. Další položkou v deklaraci atributu je zápis jeho typu a jako poslední se uvádí jeho hodnota.

```
<!ELEMENT SBIRKA (CD+)>
<!ELEMENT CD (NAZEV, INTERPRET, ROK, CENA, DELKA,
   POCETPISNI, ZANR, LABEL)>
<!ELEMENT NAZEV (#PCDATA)>
<!ELEMENT INTERPRET (#PCDATA)>
<!ELEMENT ROK (#PCDATA)>
<!ELEMENT CENA (#PCDATA)>
<!ATTLIST CENA MENA CDATA #REQUIRED>
<!ELEMENT DELKA (#PCDATA)>
<!ATTLIST DELKA JEDNOTKA CDATA #REQUIRED>
<!ELEMENT POCETPISNI (#PCDATA)>
<!ELEMENT ZANR (#PCDATA)>
<!ELEMENT LABEL (#PCDATA)>
```

Obr. 2 – Příklad DTD seznamu CD

Na příkladu je vidět, že všechny elementy kromě CD obsahují text, což je patrno ze zápisu „(#PCDATA)”. Atributy mají typ CDATA, to znamená, že mohou jako hodnotu obsahovat libovolný text. Zápis „#REQUIRED“ znamená, že zadání atributu je povinné.

2.4.3 Připojení DTD k dokumentu XML

Pokud máme vytvořený DTD, musíme ho připojit k našemu dokumentu XML. To je celkem jednoduché. Stačí do hlavičky XML dokumentu (tedy do prostoru mezi deklaraci XML a otevírací tag kořenového elementu) vložit celé námi vytvořené DTD pomocí tzv. DOCTYPE.

```
<!DOCTYPE SBIRKA [
        <!ELEMENT SBIRKA (CD+)>
                .
                .
                . ]>
```

Obr. 3 – Vložení celého DTD do XML souboru

Tento způsob je sice možný, ale v praxi se příliš nepoužívá. Pokud totiž DTD vložíme přímo do XML dokumentu, nelze ho pak sdílet s ostatními dokumenty a zbytečně tak narůstá jeho objem. Elegantnější způsob je uložení DTD do samostatného souboru a připojení pomocí odkazu.

```
<!DOCTYPE SBIRKA SYSTEM "cd.dtd">
```

Odkaz v XML souboru na externí DTD

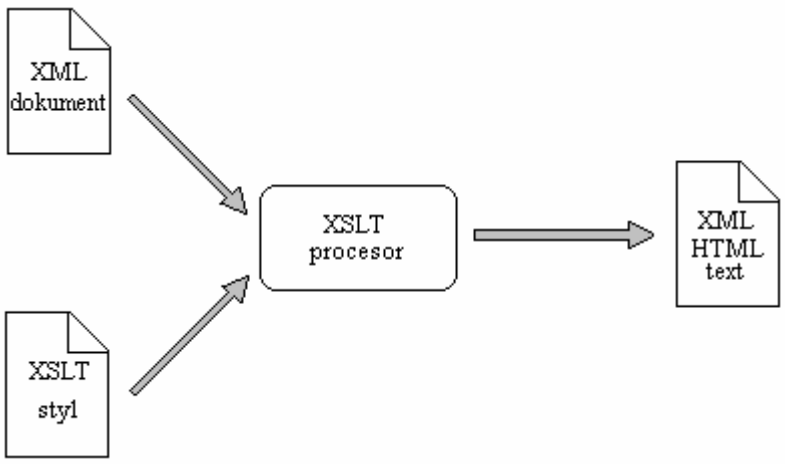
3. Tvorba stylů XSLT

3.1 Základní principy XSLT

Základní myšlenkou, na které staví většina značkovacích jazyků včetně XML a SGML, je důsledné oddělení obsahu dokumentu od jeho vzhledu. Značky použité v XML dokumentu vyznačují význam jeho jednotlivých částí. O tom, jak se konkrétní údaj zobrazí na obrazovce nebo vytiskne na tiskárně, samotný jazyk XML nic neříká. Můžeme si však odděleně vytvořit definici vzhledu jednotlivých elementů, které se říká styl.

S jeho pomocí je pak zobrazení dokumentu velice snadné. Stačí mít k dispozici aplikaci, která umí číst XML dokumenty a rozumí použitému stylovému jazyku. Dnes se nejčastěji používají dva – kaskádové styly CSS (Cascading Style Sheets) a XSL. My se budeme zabývat stylem XSL, který se používá při práci s XSLT procesory. Když vznikal jazyk XSL, umožňoval stejně jako CSS, definovat vzhled jednotlivých elementů, způsob jejich zarovnání, velikost a styl písma, barvy apod. Kromě toho jej však šlo použít i k takovým věcem, jako je automatické generování obsahu, číslování obrázků, kapitol apod. Postupně se ukázalo, že XSL má sloužit ke dvěma poměrně odlišným věcem – k transformaci XML dokumentů a k definici vzhledu jejich formátování, Během příprav standardu XSL z něj proto byla vyřazena jeho část sloužící k transformaci dokumentů, pro kterou se používá název XSLT (XSL Transformations). Pomocí XSLT lze vytvářet styly, které definují, jak se XML dokumenty mají převádět do formátu HTML, do XML dokumentů s jinou strukturou nebo do obyčejných textových souborů. Zejména možnost konverze do HTML je dnes hojně využívána, protože většina prohlížečů si zatím se samotnými XML dokumenty neporadí. Druhá část XSL, která slouží k přesnému popisu vzhledu dokumentu, se jmenuje XSL FO (formátovací objekty).

Významnou roli hraje XSLT také pro přenositelnost dat. Umožňuje totiž převést dokument sestavený podle jednoho DTD na jiné DTD a tak např. předávat data mezi aplikacemi.



Obr. 4 – Princip zpracování XML dokumentu pomocí XSLT stylu

3.2 Stromy a uzly

Pracujeme-li s XSLT, neuvažujeme již v terminologii dokumentu, ale spíše v terminologii stromových struktur. Stromová struktura reprezentuje veškerý obsah dokumentu. Tento obsah je považován za množinu uzlů (elementů, atributů atd.) uspořádaných do určité hierarchie. Toto uspořádání se v XSLT řídí doporučením konsorcia W3C pro jazyk XPath. Z pohledu XSLT jsou tedy dokumenty určitými stromy sestavenými z uzlů. XSLT rozpoznává sedm typů uzlů.

3.2.1 Kořenový uzel

Kořenový uzel je uveden na samém začátku dokumentu. Tento uzel reprezentuje v interakci s XSLT celý dokument. Jeho potomkem je vždy kořenový element dokumentu a případně další uzly (komentáře, instrukce pro zpracování apod.).

3.2.2 Uzel elementu

Skládá se z části dokumentu ohraničeného počáteční a koncovou značkou. Může to být i prázdná značka elementu. Název uzlu odpovídá názvu elementu. Každý element tvoří samostatný uzel stromu. Hierarchie elementů v dokumentu přitom odpovídá hierarchii uzlů ve stromové reprezentaci. Potomky uzlu mohou být další elementy, textové uzly, komentáře nebo instrukce pro zpracování. K uzlu mohou být připojeny atributy a jmenné prostory a hodnota uzlu je spojení všech textových uzlů – potomků – v pořadí výskytu v dokumentu.

3.2.3 Uzel atributu

Obsahuje hodnotu atributu po nahrazení všech znakových entit a po oseknutí ohraničujících mezer. Každý atribut je ve stromu připojen k odpovídajícímu elementu, ale není chápán jako jeho potomek. Avšak element, ke kterému je atribut připojen, je chápán jako jeho rodič. Název uzlu odpovídá názvu atributu, stejně jako hodnota uzlu odpovídá hodnotě atributu.

3.2.4 Textový uzel

Tento uzel obsahuje posloupnost znaků, tj. text typu PCDATA. Textové uzly jsou v XSLT implicitně normalizovány, což znamená, že sousedící textové uzly jsou slučovány. Textové uzly odpovídají textovému obsahu elementů. Automaticky jsou v nich nahrazeny skutečným textem výskyty všech entit – jak znakových, tak i interních či externích textových. Textový uzel nemá název ani nemůže mít potomky. Vždy se jedná o listový uzel. Všechny konce řádků jsou převedeny na jeden znak LF (
).

3.2.5 Uzel s instrukcí pro zpracování

Vždy se jedná o listový uzel, uzel nemůže mít potomky. Názvem uzlu je cíl instrukce. Pro každou instrukci se ve stromu vytvoří samostatný uzel, který obsahuje jak cíl, tak i samotný obsah instrukce bez značek <? a ?>. Výjimku tvoří deklarace XML „<?xml version=”1.0”?>“, která není instrukcí pro zpracování, přestože tak vypadá. Tento uzel je procesory XSLT automaticky přeskakován.

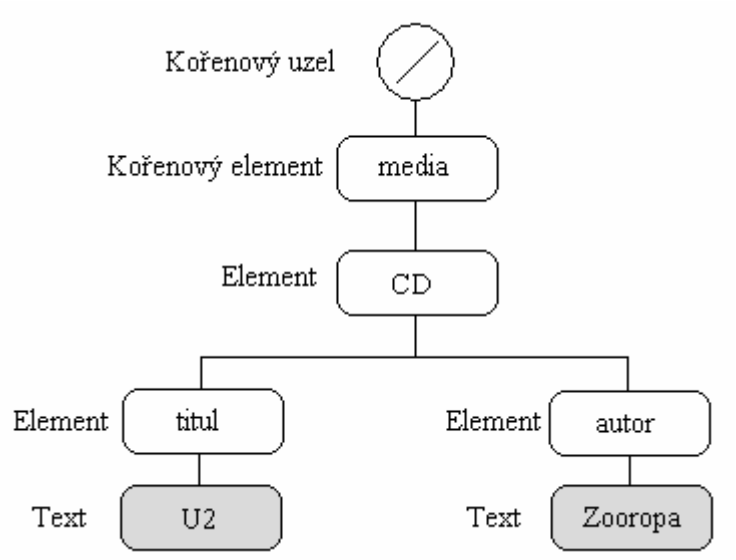
3.2.6 Uzel komentáře

Obsahuje text komentáře bez komentářových značek <!-- a -->. Pro každý komentář se ve stromu vytvoří odpovídající uzel. To je užitečné, pokud komentáře obsahují nějaké instrukce ovlivňující zpracování. Elegantnější způsob je však zařazení těchto příkazů do dokumentu pomocí instrukcí pro zpracování. Vždy se jedná o listový uzel. Tento uzel nemá název ani potomky.

3.2.7 Uzel s definicí jmenného prostoru

Uzel nese informaci o jmenném prostoru, který je deklarován. Deklarace jmenného prostoru se dědí, a proto je i tento uzel přítomný na potomcích uzlu, který odpovídá elementu s deklarací jmenného prostoru. Uzel je pouze připojen k elementovému uzlu a není chápán jako jeho potomek. Naproti tomu element,

ke kterému je uzel připojený, je chápán jako jeho rodič. Opět se vždy jedná o listový uzel. Název uzlu odpovídá prefixu jmenného prostoru. Hodnotou uzlu je jeho URI (Uniform Resouce Identifier) adresa.



Obr. 5 – Příklad stromové struktury

Na obrázku je znázorněna stromová struktura následujícího XML dokumentu, jak je chápána v terminologii procesoru XSLT.

```
<?xml version=”1.0” encoding=”windows-1250”?>
<media>
        <CD>
                <titul>U2</titul>
                <autor>Zooropa</autor>
        </CD>
</media>
```

Obr. 6 – XML dokument pro strom z obr. 5

Uvedený stromový diagram však není úplně kompletní, záměrně jsem vynechal jeden typ uzlu, kterým je speciální textový uzel, který může obsahovat jenom prázdná místa. Tento typ uzlu se stará o to, aby byl XML dokument upraven tak, jak ho vidíme. Tedy o mezery, znaky nového řádku, znaky posuvu řádku a tabelátory. Procesory XSLT tyto uzly implicitně uchovávají a předávají do výsledného dokumentu.

3.3 Připojení Stylu k XML dokumentu

Aby bylo zajištěno to, že se při zobrazování dokumentu bude používat určitý styl, je třeba to aplikacím sdělit. K tomu slouží instrukce pro zpracování *xml-stylesheet*. Ta musí dále obsahovat URL adresu, na které je uložen styl a typ použitého stylového jazyka.

```
<?xml-stylesheet href="styl.xsl" type="text/xsl"?>
```

Příklad připojení stylu

Tento zápis se musí uvádět vždy na začátku dokumentu. Tato instrukce však není součástí doporučení pro XSLT a většinou samostatných XSLT procesorů není podporována.

3.4 Základní elementy – instrukce pro zpracování

Element xsl:stylesheet

Jedná se o kořenový element, který je povinný a musí obsahovat atribut *version* určující použitou verzi jazyka XSLT. Dalším atributem je deklarace jmenného prostoru zapisující se jako „xmlns:xsl=“http://www.w3c.org/1999/XSL/Transform“, ve kterém nesmí být chyba, a to ani ve velikosti písmen.

Element xsl:template

Pomocí toho elementu se tvoří šablony, které jsou základním stavebním prvkem dokumentu XSLT. Povinným atributem je atribut *match,* který určuje prvek ze vstupního dokumentu. V těle šablony je pak popsáno, co se má s vybraným prvkem udělat.

Element xsl:apply-templates

Tento element umožňuje přechod na přímé potomky aktuálního elementu a hledání jim odpovídajících šablon.

Element xsl:value-of

Tento element umožňuje zapsat do výstupního dokumentu buď veškerý textový obsah prvku ze zdrojového dokumentu, nebo se do něj mohou vkládat výsledky vyhodnocení dalších výrazů. Element nebo atribut, kterého se to týká, se určuje pomocí výrazu XPath uvedeného v hodnotě povinného atributu *select.*

Element xsl:for-each

Používá se v XSLT dokumentech k vytvoření cyklického opakování příkazů. Kdykoliv procesor v dokumentu narazí na tento element, vykoná příkazy uvedené uvnitř těla tohoto elementu. A to pro každý prvek, který odpovídá výrazu XPath zapsaném v hodnotě povinného atributu *select.*

Element xsl:sort

Pomocí tohoto elementu se prvky ze vstupního dokumentu nějakým způsobem třídí. Způsob třídění se ovlivňuje pomocí pěti nepovinných atributů. Pokud se žádný z nich nepoužije, jako klíč k setřídění bude sloužit aktuální prvek a výchozí prvky budou seřazeny vzestupně podle abecedy. Tento element se používá v těle příkazů *xsl:apply-templates* a *xsl:for-each*.

Element xsl:text

Využijeme pokud potřebujeme do nějakého místa ve výstupním dokumentu zapsat určitý text, který není obsažen v datech vstupního dokumentu.

Element xsl:if

Tento element umožňuje vytvořit v transformačních dokumentech podmínky. Příkazy uvnitř tohoto elementu budou vykonány pouze v případě, že je splněna podmínka definovaná pomocí výrazu XPath, který je uveden v hodnotě povinného atributu *test.*

Element xsl:choose

Použití toho elementu je výhodné, pokud chceme vytvořit test obsahující více podmínek. Užívá se v kombinaci s elementy *xsl:when* a *xsl:otherwise*. Jediným atributem v tomto mechanismu je povinný atribut test u elementu *xsl:when*. Funguje stejným způsobem jako u elementu *xsl:if*, ale s jedním rozdílem. Pokud je pro určitý prvek nějaká podmínka splněna, jiné se v jeho případě již neprověřují.

Element xsl:output

Tento element se používá na určení výstupního formátu. Je to prázdný element, který musí být umístěn ihned za počátečním tagem elementu *xsl:stylesheet*. Formát výstupního dokumentu lze definovat pomocí atributu *mehtod*, ten může nabývat hodnot „xml“ (dokument xml), „html“ (dokument HTML) nebo „text“ (textový dokument).

Element xsl:atribute

Pomocí tohoto elementu se na výstupu vytvářejí nové atributy. Jeho povinným atributem je *name*, který určuje jméno nového atributu na výstupu, jeho hodnotu určuje obsah elementu.

Element xsl:element

Tento element má podobnou funkci jako předchozí, ale na rozdíl od něj za běhu vytváří elementy. Opět má povinný atribut *name*.

Element xsl:copy

Tento element vytváří na výstupu kopii aktuálního elementu, ale bez jeho případného obsahu i atributů. Toho lze využít, pokud bychom měli vstupní dokument zjednodušit co se týče množství použitých elementů, ale nikoliv jejich jmen.

Element xsl:copy-of

Na rozdíl od předchozího prvku vytváří na výstupu přesnou kopii elementu určeného pomocí povinného atributu *select*, tedy včetně jeho obsahu (text i vnořené elementy) a případných atributů.

3.5 Jmenné prostory

Jmenné prostory slouží k rozlišení elementů a atributů se shodnými jmény v případech, kdy by mohlo dojít ke konfliktům. Jména elementů a atributů se pak skládají ze dvou částí, z jmenného prostoru a z lokálního názvu. Jmenný prostor je určen URI identifikátorem, který by měl být v celosvětovém měřítku jedinečný. Například všechny instrukce XSLT procesoru patří do společného jmenného prostoru „http://www.w3.org/1999/XSL/Transform“. Touto URI je tedy určen význam XML elementů či atributů a slouží pouze jako identifikátor (nic konkrétního se na ní

nevyskytuje) Pokud aplikace jmenný prostor nezná, měla by dotyčné elementy či atributy ignorovat.

Pro zkrácení zápisu se při deklaraci jmenného prostoru pro XSLT vytvoří prefix, který jmenný prostor zastupuje. A to pomocí speciálního atributu *xmlns*.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Příklad deklarace jmenného prostoru

Na obrázku je příklad deklarace jmenného prostoru pro prefix „xsl“, který se používá ve stylech XSLT. Ve skutečnosti můžeme pro definici elementů XSL použít jakýkoli prefix jmenného prostoru, nebo nemusíme použít žádný.

4. Jazyk XPath

Základním stavebním prvkem XSLT transformací je jazyk XPath, ten umožňuje uživatelům odkazovat se na určité oddíly dokumentů XML. Je to jazyk pro adresování různých částí dokumentů. XPath je nástrojem k určení toho, s jakou částí zdrojového dokumentu chceme vlastně pracovat. Aby mohl vykonávat tyto funkce, poskytuje tento jazyk základní prostředky pro manipulaci s řetězci, čísly a booleovskými hodnotami. XPath používá kompaktní syntaxi, odlišnou od XML, která umožňuje užití XPath v adresách URI a v hodnotách atributů XML.

Další jeho vlastností je to, že operuje s abstraktní logickou strukturou dokumentu XML, nikoli s jeho povrchovou syntaxí. Název „XPath“ vznikl s ohledem na zápis cesty URI používané pro navigaci uvnitř hierarchické struktury dokumentu XML. Bavíme se o XPath 1.0, jehož finální doporučení bylo vydáno 16. listopadu 1999. Rozpracovaná je však již verze 2.0, která je v současné době ve fázi „kandidátního doporučení“. XPath 2.0 přináší následující novinky:

- změna pohledu na hodnoty vrácené XPath výrazem, vše jsou sekvence
- zavádí podmíněné výrazy a cykly
- zavádí možnost uživatelských funkcí

4.1 Datové typy jazyka XPath

Jazyk XPath rozlišuje pět datových typů a těmi jsou množina uzlů, booleovské hodnoty, čísla, řetězce a části výsledného stromu. Poslední jmenovaný se týká dílčích částí stromu, které lze přiřazovat proměnným XSLT. Tento datový typ byl však vyloučen již z pracovního konceptu XSLT 1.1, z čehož plyne, že nebude ani součástí specifikace XSLT 2.0. Proto se o něm nebudu dále zmiňovat.

4.1.1 Datový typ Množina uzlů

Tento datový typ může obsahovat libovolný počet uzlů, jeden jediný, ale i žádný. Vzhledem k tomu, že základním posláním jazyka XPath je vyhledávat jednotlivé oddíly uvnitř dokumentů, jsou výrazy, které vracejí množiny uzlů, nejoblíbenější. Mezi základními funkcemi je třeba upozornit zejména na funkce *name* a *local-name*, které poskytují velmi dobrý způsob, jak určit název aktuálního elementu. Například, je-li aktuálním elementem element <CD> , vrací funkce *local-name* hodnotu „CD“.

4.1.2 Číselný datový typ

V jazyce XPath jsou čísla uchovávána ve formátu s dvojitou přesností v pohyblivé řádové čárce (double floating point format). Jedná se vlastně o 64bitový formát. K manipulaci s čísly je k dispozici několik operátoru, těmi jsou:

- + sčítání
- - odčítání
- * násobení
- div dělení
- mod zbytek po dělení

4.1.3 Datový typ Řetězec

Řetězce v jazyku XPath se implicitně skládají ze znaků v kódování Unicode. Pro práci s řetězci je určeno mnoho funkcí, z nichž asi nejpoužívanější jsou tyto:

- string() – převede libovolný objekt na řetězec
- concat() – spojí několik řetězců do jednoho
- contains() – otestuje, zda řetězec obsahuje hledaný řetězec
- string-lenght() – vrací délku (počet znaků) řetězce

4.1.4 Datový typ Boolean

Booleovské výrazy v jazyku XPath (stejně jako v jiných jazycích) vracejí buď pravdu nebo nepravdu. Používají se obvykle v predikátech. Za pravdu je považováno jakékoli nenulové číslo a jakýkoliv řetězec, zatímco nula a prázdný řetězec jsou považovány za nepravdu. Lze používat následující operátory jazyka XPath:

- != znamená „nerovná se“
- < znamená „je menší než“
- <= znamená „je menší nebo se rovná“
- = znamená „rovná se“
- > znamená „je větší než“
- >= znamená „je větší než nebo se rovná“

K propojení booleovský klauzulí můžeme použít klíčová slova *and* nebo *or*. K převrácení logického významu výrazu pak *not*.

4.2 Výrazy pro výběr části dokumentu pomocí cesty

Uzly se ze stromu dokumentu vybírají na základě námi stanovených kritérií a vzájemné pozice uzlů. Výrazy přitom mohou strom dokumentu prohledávat z několika míst. Buď od aktuálního uzlu, přičemž použijeme relativní cestu, nebo od kořene, kdy použijeme absolutní cestu. Ta začíná znakem „/“, jež označuje přístup k celému dokumentu jako k celku. Pokud v nějakém výrazu použijeme zápisu „//“, znamená to, že přistupujeme do všech možných nižších úrovní ve „stromu“. Poslední možností je prohledávat z určeného místa, na které se dostaneme například funkcí *id()*.

Cesta se může skládat z několika částí, které se oddělují znakem „/“ Jednotlivé části cesty jsou po krocích procházeny zleva doprava a postupuje se tak stromem dolů. Počet uzlů v jednom kroku snižují jeho jednotlivé komponenty – osa, test uzlu a predikát. Znak „*” vybere všechny elementy, které jsou určené předcházejícím výrazem. Pokud bychom chtěli vybrat nějaký atribut v dokumentu, můžeme ve výrazu XPath využít znaku „@”. Například zápis *//@** umožňuje oslovit libovolný atribut na jakékoliv úrovni v dokumentu. Každá část cesty se skládá z několika komponent:

- identifikátoru osy – ten určuje, ve kterém směru se budeme od aktuálního uzlu pohybovat
- testu uzlu – umožňuje vybírat jen některé uzly na základě jejich typu a názvu
- predikátů – vybrané uzly můžeme dále filtrovat pomocí podmínek, které jsou testována pro každý uzel

4.2.1 Identifikátory osy

Pohyb po určité ose stromu dokumentu se zapisuje pomocí identifikátoru osy následovaného dvěma dvojtečkami. Implicitní osou je *child::*. Nejpoužívanějšími identifikátory jsou:

- ancestor:: – Rodič aktuálního uzlu a všichni jeho další předci až ke kořenu stromu dokumentu. Uzly jsou uspořádány v opačném pořadí než v dokumentu.
- ancestor-or-self:: – Aktuální uzel a všichni jeho další předci až ke kořenu stromu dokumentu. Uzly jsou uspořádány v opačném pořadí než v dokumentu.
- Attribute:: – Uzly odpovídající všem připojeným atributům, pořadí není nijak pevně definováno.Tento identifikátor lze nahradit znakem „@”

- child:: – Všechny děti aktuálního uzlu ve stejném pořadí jako v dokumentu.
- descendant:: – Všichni potomci aktuálního uzlu ve stejném pořadí jako v dokumentu.
- descendant-or-self:: – Aktuální uzel a všichni jeho potomci ve stejném pořadí jako v dokumentu.

```
//JMENO/ancestor::*
        Osa "ancestor" obsahuje  všechny předky kontextového uzlu
<CD>
        <NAZEV></NAZEV>
        <INTERPRET>
                <JMENO></JMENO>
                <PRIJMENI></PRIJMENI>
        </INTERPRET>
        <CENA MENA="Kc"></CENA>
        <DELKA JEDNOTKA="minut"></DELKA>
</CD>
```

Obr. 7 – Příklad použití identifikátoru ancestor

```
/CD/INTERPRET/descendant-or-self::*
        Osa "descendant-or-self" obsahuje kontextový uzel a všechny jeho potomky.
<CD>
        <NAZEV></NAZEV>
        <INTERPRET>
                <JMENO></JMENO>
                <PRIJMENI></PRIJMENI>
        </INTERPRET>
        <CENA MENA="Kc"></CENA>
        <DELKA JEDNOTKA="minut"></DELKA>
</CD>
```

Obr. 8 – Příklad použití identifikátoru descendant-or-self

//Attribute::MENA lze zapsat i jako **//@MENA**

Vybere všechny atributy mena.

```
<CD>
        <NAZEV></NAZEV>
        <INTERPRET>
                <JMENO></JMENO>
                <PRIJMENI></PRIJMENI>
        </INTERPRET>
        <CENA MENA="Kc"></CENA>
        <DELKA JEDNOTKA="minut"></DELKA>
</CD>
```

Obr. 9 – Příklad použití identifikátoru Attribute

4.2.2 Testy uzlu

Test uzlu umožňuje vybrat jen některé uzly na vybrané ose. Testovat můžeme název uzlu a jeho typ. Můžeme rovněž použít znak „*", který zastupuje uzel s libovolným názvem. Pokud chceme vybírat jiné uzly než elementy, atributy a jmenné prostory, můžeme použít ještě následující testy:

- processing-instruction(*jméno*) – vybere všechny instrukce pro zpracování s daným jménem
- comment() – vybere všechny komentáře
- text() – vybere všechny textové uzly
- node() – vybere všechny uzly bez ohledu na jejich typ

CD/INTERPRET/*

Výraz vybere všechny elementy, které jsou přímými potomky CD/INTERPRET

```
<CD>
        <NAZEV></NAZEV>
        <INTERPRET>
                <JMENO></JMENO>
                <PRIJMENI></PRIJMENI>
        </INTERPRET>
        <CENA MENA="Kc"></CENA>
        <DELKA JEDNOTKA="minut"></DELKA>
</CD>
```

Obr. 10 – Příklad použití znaku „" při výběru cesty*

4.2.3 Predikáty

V každé části cesty můžeme použít jeden nebo více predikátů, které sníží počet vyhovujících uzlů. Predikáty se zapisují do hranatých závorek a vyhodnocují se pro každý uzel, který zatím vyhovuje cestě. Pokud je pro uzel predikát pravdivý, uzel zůstává ve množině uzlů, které vyhovují cestě. Při vyhodnocování predikátu se mění aktuální uzel na uzel, pro který se testuje predikát. Predikát se vyhodnocuje jako logická hodnota, která filtruje uzly. Výraz použitý v predikátu může vracet různé typy a podle toho se pak vyhodnotí logická hodnota predikátu.

- číselná hodnota – Pokud predikát obsahuje výraz, který vrací číslo, chápe se jako pozice v množině uzlů na ose, po které se v dané části cesty pohybujeme.
- množina uzlů – Jako výraz v predikátu můžeme použít XPath výraz, který vrací množinu uzlů. Pokud je množina uzlů neprázdná, má predikát hodnotu true, v opačném případě false.
- ostatní výrazy – Ostatní výrazy jsou převedeny na logickou hodnotu a jejich výsledek je i hodnotou predikátu pro daný uzel.

//CENA[@MENA='Kc']

Vyber element CENA, jehož atribut “mena“ má hodnotu “Kc“

```
<CD>
        <CENA MENA="Kc"></CENA>
        <CENA MENA="Euro"></CENA>
        <DELKA JEDNOTKA="minut"></DELKA>
</CD>
```

//*[string-length(name()) = 5]

Tento výraz vybere všechny elementy, které mají pětipísmenné jméno

```
<CD>
        <NAZEV></NAZEV>
        <INTERPRET>
                <JMENO></JMENO>
                <PRIJMENI></PRIJMENI>
        </INTERPRET>
        <CENA mena="Kc"></CENA>
        <DELKA JEDNOTKA="minut"></DELKA>
</CD>
```

Obr. 11 – Příklady použití predikátů

5. XSLT procesory

XSLT procesory jako takové jsou programy, které vezmou dokument ve formátu XML, aplikují na něj XSLT styl, na jehož základně na výstup dodají transformovaný dokument. Pokud chceme dále použít XSL-FO procesor, je třeba, aby výstupem XSLT procesoru byl XML dokument ve formátu XSL-FO, který se pak použije jako vstup do formátovacího procesoru

XSLT procesory můžeme rozdělit na ty, které jsou integrovány v prohlížečích na samostatně volně šiřitelné a komerční.

5.1 XSLT procesory integrované v prohlížečích

Drtivá většina internetových prohlížečů dnes již obsahuje nějaký XSLT procesor a podporuje tak XSLT a XML. Zde je jejich stručný přehled:

- Firefox 2.0 – má v sobě integrovanou podporu formátů XML s implemetací XSLT 1.0
- Netscape 8 – pracuje na podobném enginu jako firefox, má tedy i stejnou podporu
- Opera 9 – tento prohlížeč podporuje XSLT 1.0 a XPath 1.0. S výjimkou elementu <namespace-alias>, atributů *nan*, *per-mille,* znaků „mínus“ a „procento“ u elementů s desítkovou hodnotou. Atribut *case-order* u elementu *sort* není též podporován, stejně jako XSLT:FO.
- Internet Explorer 7 – tento prohlížeč od Microsoftu má plnou podporu XML, včetně jmenných prostorů, stylů v CSS a XSLT. Podporuje parser MSXML 6.0, který implementuje standardy W3C pro XSLT 1.0 a XPath 1.0.

5.2 Volně šiřitelné XSLT procesory

Jak bylo již uvedeno, druhou částí jsou volně šiřitelné procesory. Jedná se v podstatě o samostatné programy napsané buď v jazyce C/C++, nebo v Javě. Pro procesory pracující na bázi Javy je potřeba mít nainstalované Java prostředí. Mezi nejznámější, a tudíž i nejpoužívanější, patří procesory Xalan, Saxon, xsltproc, Sablotron a XT.

5.2.1 Xalan

Tento procesor je vyvíjen a spravován organizací Apache. Původní kód však pochází z laboratoří IBM. Procesor je převážně používán ve verzi pro Javu, ale existuje i verze psaná v C++. Neustále se vyvíjí a jeho poslední vydání je Xalan-Verison 2.7.0. To má v sobě zabudovanou částečnou podporu EXSLT (rozšíření snažící se standardizovat funkce XSLT procesorů pro lepší přenositelnost stylových dokumentů). Mezi jeho vlastnosti patří:

- je dostupný včetně zdrojových textů
- úplně implementuje XSLT 1.0 a XPath 1.0. Možnost definice vlastních funkcí v libovolném jazyce, který podporuje rozhraní BSF (Bean Scripting Framework) – Java, JavaScript, VBScript, ReXX, Python, Perl atd.
- C++ verze na vstupu podporuje mnoho kódování včetně těch českých (používá se ICU knihovna), po překompilování lze využít různá kódování i na výstupu
- ve verzi 2.6.2 se vyskytly problémy s generováním rejstříků
- může být použit jako servlet transformující XML do HTML a poskytující výsledek dalším klientům
- transformace mohou být zřetězeny
- může pracovat s libovolným XML parserem, jako např Xerxes-Java
- dostupný včetně zdrojových textů

5.2.2 Saxon

Jedná se o volně šiřitelný procesor, který vyvinul Michale H. Kay. Existuje ve verzi pro Javu, ale i pro platformu .NET. Poslední verze Saxon 8.7.1 je vydávána ve dvou variantách, Saxon-B a Saxon-SA. Obě dvě implementují XSLT 1.0 a 2.0 a navíc i jazyk XQuery 1.0. Specifikace XSLT 2.0/XQuery 1.0 rozlišuje mezi dvěma možnými implementacemi – základním procesorem a procesorem podporujícím jazyk XML Schéma (náhrada a rozšíření DTD).

Hlavní rozdíly mezi oběma variantami jsou ty, že Saxon-B zůstal „open-source“ projektem, naproti tomu Saxon-SA je komerčním produktem firmy Saxonica, kterou založil právě Michale Kay. Dále se odlišují tím, že Saxon-SA podporuje právě jazyk XML Schéma, má schopnost zachycovat dynamické chyby, umí formátovat data a čísla

a podporuje více funkcí než Saxon-B.Mezi základní vlastnosti procesoru Saxon-B patří to, že podporuje:

- práci s větším počtem výstupních dokumentů
- množinové operace s uzly
- víceprůchodové zpracování dokumentů
- možnost definice vlastních funkcí
- kombinace javového kódu s XSLT
- návrh XPath 2.0
- změny v obsahu proměnných
- kódování iso-8859-2 a windows-1250
- možnost na vstupu použít libovolný parser

5.2.3 Xsltproc

Tento procesor je postavený nad knihovnou *libxslt*. Tato knihovna je napsaná v jazyku C a je vyvíjena jako součást projektu Gnome, tudíž je šířena pod GPL licencí. Je v ní částečná implementace EXSLT rozšíření. Dále má velice dobrou podporu XInclude, což je nástroj umožňující např. složení XML dokumentu z několika dílčích souborů a vkládání XML fragmentů nebo textových souborů. Jednou z hlavních předností je to, že xsltproc je jediný nástroj, který umožňuje transformaci do XHTML formátu. Poslední verze pro Windows byla vydána 6. ledna 2005, pro Unix pak 29. listopadu 2006. Mezi další vlastnosti patří:

- plná implementace XSLT 1.0 a XPath 1.0
- dostupný včetně zdrojových textů
- možnost definice vlastních rozšíření v C
- s knihovnou *iconv* podporuje velké množství kódování
- dominantní XML/XSLT knihovna v unixovém světě, dostupná i pro Windows

5.2.4 XT

Tento procesor pracuje na bázi Javy, k transformacím používá parser XP. Byl vyvinut Jamesem Clarkem, ale v současné době ho spravuje Bill Linsdey. Lze s ním použít i jiné parsery, ale to není doporučeno.

Téměř úplně implementuje XSLT 1.0 a XPath 1.0. Poslední verze toho procesoru, která obsahuje mimo jiné i doplňkové funkce EXSLT, byla vydána 6. prosince 2005. Tento procesor mimo jiné nezvládá zpracovat dokumenty s kódováním windows-1250, tedy ani českou diakritiku. Mezi další specifika XT patří:

- podporuje více vstupních dokumentů
- několik množinových operací s uzly
- možnost definice vlastních funkcí
- upravená verze podporuje kódování windows-1250 a iso-8859-2. Katalogové soubory, které umožňují snadné přemapování veřejných a systémových identifikátorů. A výstupní metodu XHTML, která umožňuje generování XHTML kódu kompatibilního i se staršími prohlížeči.
- může být použit jako servlet
- podporuje více výstupních dokumentů
- vývoj byl již zastaven, šlo pouze o ověřovací implementaci standardu

5.2.5 Sablotron

Jedná se o procesor, vyvinutý českou firmou Ginger Alliance, s.r.o. Je to současně první XSLT procesor napsaný v jazyce C++. Jedná se o procesor plně implementující specifikaci XSLT 1.0, XPath 1.0 a DOM Level2. Sablotron využívá Expat XML parser vyvinutý James Clarkem, který je v současné době ve verzi 2.0.

5.2.6 Jd.xslt

Je procesor vytvořený Johannesem Döblerem. Stejně jako většina předchozích i tento pracuje na platformě Java. Jeho poslední vydání ze dne 13. května 2003 implementovalo XPath 1.0 a pracovní koncept XSLT 1.1, který byl, stejně jako další vývoj tohoto procesor, zastaven. Domovské stránky tohoto projektu jsou nedostupné, stejně jako sám autor Johannes Döbler.

6. Nástroje pro testování XSLT procesorů

6.1 Hardware pro testování

Testování XSLT procesorů jsem prováděl na počítači běžícím pod operačním systémem Windows XP, SP2. Konfigurace:

- Procesor – Intel Pentium 4-2.4 GHz, 512 kB L2 Cache
- Základní deska – Intel D845PESV
- Operační paměť – 1 GB DDR-SDRAM 333 MHz

6.2 Software pro testování

6.2.1 Cygwin

Tento software získaný z http://www.cygwin.com/ slouží k emulaci unixového prostředí pod Windows. Jedná se o rozsáhlý balík, který obsahuje mnoho součástí, které jsou běžné pro unixové distribuce. Důvod, proč jsem jej použil při testování byl ten, že mimo jiné obsahuje i funkci *time*. Ta ve Windows nemá svojí obdobu a díky ní jsem mohl měřit jednotlivé časy transformací.

6.2.2 Java 2 Runtime Environment SE

Jedná se o prostředí Javy, které bylo potřeba ke spouštění Java procesorů Saxon, Xalan-J a XT. Naše verze byla konkrétně Java Runtime Environment, SE v1.4.2_04. Tu lze získat například z http://java.sun.com/javase/.

6.2.3 DocBook

Poslední verzi DocBook 1.71.0 jsem stáhl z domovské stránky tohoto projektu (http://sourceforge.net/projects/docbook). Jedná se o systém sloužící především pro tvorbu dokumentací, knih, článků, prezentací nebo webů. Je založen na XML a dají se pomocí něj dělat snadné konverze do HTML, PDF, RTF, nebo jiných formátů. Mimo jiné obsahuje i složité styly, které jsem použil při testování.

6.3 XSLT procesory

6.3.1 Sablotron

Pro mé účely jsem z internetových stránek společnosti Ginger Aliance (http://www.gingerall.org/) stáhl poslední verzi procesoru, tedy 1.0.3. Po rozbalení archivu jsem získal v adresáři *bin* soubor sabcmd.exe. Pomocí něj se spouští procesor.

```
procesory/sablotron/bin/sabcmd.exe stylesheet input file output file
```

Způsob spouštění transformace

6.3.2 Saxon-B 8.8

Abych mohl testovat tento procesor, stáhl jsem z jeho domovských stránek (http://saxon.sourceforge.net/) poslední verzi. Je jí Saxon-B 8.8. Ten je distribuován jako archiv jazyka Java.

```
java -jar procesory/saxon/saxon8.jar input file  stylesheet  >output file
```

Způsob spouštění transformace

Spouští se příkazem „java -jar" Dále následuje cesta k vlastnímu archivu s procesorem (saxon8.jar), k XML dokumentu, stylu a za znakem „>" je umístění transformovaného souboru.

6.3.3 Saxon 6.5.3

Tento procesor jsem stáhl ze stejných stránek jako předchozí (http://saxon.sourceforge.net/). Jak je patrné, jedná se o jeho starší verzi, kterou jsem využil tam, kde Saxon-B 8.8 nepracoval správně. Konkrétně při testování se složitými styly. Opět se jedná o Java archiv, z čehož plyne i stejné spouštění.

6.3.4 Xalan-J

Jak již bylo uvedeno, tento procesor je vyvíjen a spravován organizací Apache, z jejíchž stránek (http://xml.apache.org/xalan-j/) jsem také stáhl jeho poslední verzi Xalan Java Version 2.7.0.

```
java -jar procesory/xalan/bin/xalan.jar -IN input file -XSL stylesheet -OUT output file
```

Způsob spouštění transformace

Z příkladu je vidět, že je tento procesor distribuován jako Java archiv (xalan.jar). Z toho plynou i stejné podmínky pro spuštění jako u předešlého procesoru. Rozdíl je v zadávání vstupních dokumentů.

6.3.5 Xalan-C

Jedná se o procesor Xalan psaný v jazyce C++. Stejně jako jeho Java verze, je i tento ke stažení na stránkách organizace Apache (http://xml.apache.org/xalan-c/). Odtud jsem tedy získal poslední verzi procesoru – Xalan-C++ version 1.10 a k jeho běhu nezbytný XML parser Xerces-C++ version 2.7.0. Po rozbalení obou archivů jsem získal soubor xalan.exe, který spouští procesor.

```
procesory/xalan-C/bin/xalan.exe -o output file input file stylesheet
```

Způsob spouštění transformace

6.3.6 Xsltproc

Tento procesor lze najít na stránkách (http://xmlsoft.org/XSLT/), kde se nachází poslední verze – libxslt 1.1.15 a další potřebné archivy. Protože je tento procesor napsán v jazyce C, není k jeho chodu třeba Java a je poskytován jako spustitelný soubor – xsltproc.exe

```
Procesory/xsltproc/bin/xsltproc.exe -o output file stylesheet input file
```

Způsob spouštění transformace

6.3.7 XT

Procesor XT jsem stáhl z ftp Jamese Clarka (ftp://ftp.jclark.com/pub/xml/). Jedná se o speciálně upravenou verzi pro win32. Je to tedy spustitelný exe soubor a má v sobě vše potřebné (včetně parseru).

```
Procesory/xt/xt.exe input file stylesheet output file
```

Způsob spouštění transformace

6.3.8 XT s úpravou pro český jazyk

Jak již bylo uvedeno, originál procesoru XT nezvládá kódování s českou diakritikou. Existuje však jeho rozšířená verze, která je volně ke stažení ze stránek jejího autora (http://www.kosek.cz/xml/xt/). Tato verze mimo jiné zvládá i kódování windows-1250.

```
java -jar procesory/xtcz/xt.jar  com.jclark.xsl.sax.Driver input file stylesheet output file
```

Způsob spouštění transformace

Tentokrát se jedná o procesor napsaný v Jave, z čehož plyne i způsob jeho spouštění. Za zmínku zde stojí použití třídy transformace „com.jclark.xsl.sax.Driver“.

7. Testování výkonu procesorů

7.1 Postup při testování

Testování výkonu procesorů jsem provedl na převodu XML do HTML. Abych zjistil, zda jsou procesory celkově rychlé či pomalé, nebo se na výkonu projevuje rychlý nebo pomalý start, bylo třeba použít různě velké vstupní XML dokumenty. Pro tyto potřeby jsem stáhl z http://wurfl.sourceforge.net/ 3,7 MB velký soubor wurf.xml, který je volně šiřitelnou databází obsahující schopnosti, vlastnosti a parametry mobilních zařízení přistupujících k WAP. Tento soubor jsem si vybral díky jeho struktuře, která ho umožňovala poměrně snadno rozdělit na malé a střední části.

Vytvořil jsem tedy dvacet středně velkých souborů o velikosti v rozmezí 50–100 kB, sedmdesát malých souborů velkých 1–15 kB a jako velký jsem nechal celý wurfl.xml. Ze stejných stránek, kde jsem stáhl zmiňovaný soubor, jsem k němu získal i stylový dokument convert_2_html.xsl, který jsem použil při testovacích transformacích.

Abych mohl porovnávat výkon jednotlivých procesorů, měřil jsem časy transformací, a to pomocí unixového příkazu *time* v cygwin. Ten pracuje tak, že se před příkaz pro spuštění měřeného programu napíše jednoduše „time“

```
time procesory/sablotron/bin/sabcmd.exe wurfl/convert_2_html.xsl stredni/s01.xml
```

Příklad použití funkce time

Program *time* pak vrátí tři hodnoty času. *Real*, která udává reálný čas, u*ser*, udávající čas uživatelského CPU a s*ys*, vypisující čas systémového CPU. Mě zajímal čas r*eal*.

```
real    0m10.288s
user    0m0.426s
sys     0m0.272s
```

Příklad výstupu funkce time

7.2 Testování na velkém souboru

Nejprve jsem testoval procesory na velkém souboru wurfl.xml. Pro každý jsem vytvořil dávkový soubor.

```
procesory/xsltproc/bin/xsltproc -o out/xprw.html wurfl/convert_2_html.xsl wurfl/wurfl.xml
```

Příklad dávkového souboru

Ten jsem pak spouštěl z příkazové řádky cygwinu pomocí příkazu *sh*, před který jsem pro měření času přidal příkaz *time.*

```
time sh xpr.bat
```

Příklad spouštění dávkového souboru

Naměřené časy transformací jednotlivých procesorů jsem zaznamenal v následující tabulce.

Tab. 1 – Naměřené časy transformací velkého souboru

	Sablotron	Saxon	Xalan-J	Xalan-C	xsltproc	XT
čas [s]	23,433	8,271	12,495	6,487	4,758	6,277

7.3 Testovaní na středně velkých souborech

Při tomto testování jsem místo wurfl.xml používal mnou vytvořené soubory s01.xml až s20.xml. Dále pak jeden dávkový soubor, do kterého jsem vložil všech dvacet transformací.

```
procesory/xsltproc/bin/xsltproc -o out/xpr.html wurfl/convert_2_html.xsl stredni/s01.xml
procesory/xsltproc/bin/xsltproc -o out/xpr.html wurfl/convert_2_html.xsl stredni/s02.xml
.
.
.
procesory/xsltproc/bin/xsltproc -o out/xpr.html wurfl/convert_2_html.xsl stredni/s20.xml
```

Příklad dávkového souboru

Spouštění dávkového souboru bylo stejné jako při testování na velkém souboru. Naměřené časy jsem opět zaznamenal do tabulky.

Tab. 2 – Naměřené časy transformací středně velkých souborů

	Sablotron	Saxon	Xalan-J	Xalan-C	xsltproc	XT
čas [s]	9,917	25,038	24,667	3,734	3,37	6,288

7.4 Testování na malých souborech

Malé soubory jsem testoval stejně jako středně velké. S tím rozdílem, že jsem používal soubory m01.xml až m70.xml a do dávkového souboru jsem místo dvaceti spuštění vložil všech sedmdesát.

```
procesory/xsltproc/bin/xsltproc -o out/xpr.html wurfl/convert_2_html.xsl male/m01.xml
procesory/xsltproc/bin/xsltproc -o out/xpr.html wurfl/convert_2_html.xsl male/m02.xml
.
.
.
procesory/xsltproc/bin/xsltproc -o out/xpr.html wurfl/convert_2_html.xsl male/m70.xml
```

Příklad dávkového souboru

Po spuštění všech šesti transformací, které bylo stejné jako v případě testování velkých a středně velkých dokumentů, jsem všechny naměřené hodnoty zanesl do následující tabulky.

Tab. 3 – Naměřené časy transformací malých souborů

	Sablotron	Saxon	Xalan-J	Xalan-C	xsltproc	XT
čas [s]	4,306	64,701	54,617	4,945	2,984	13,724

7.5 Výsledky měření

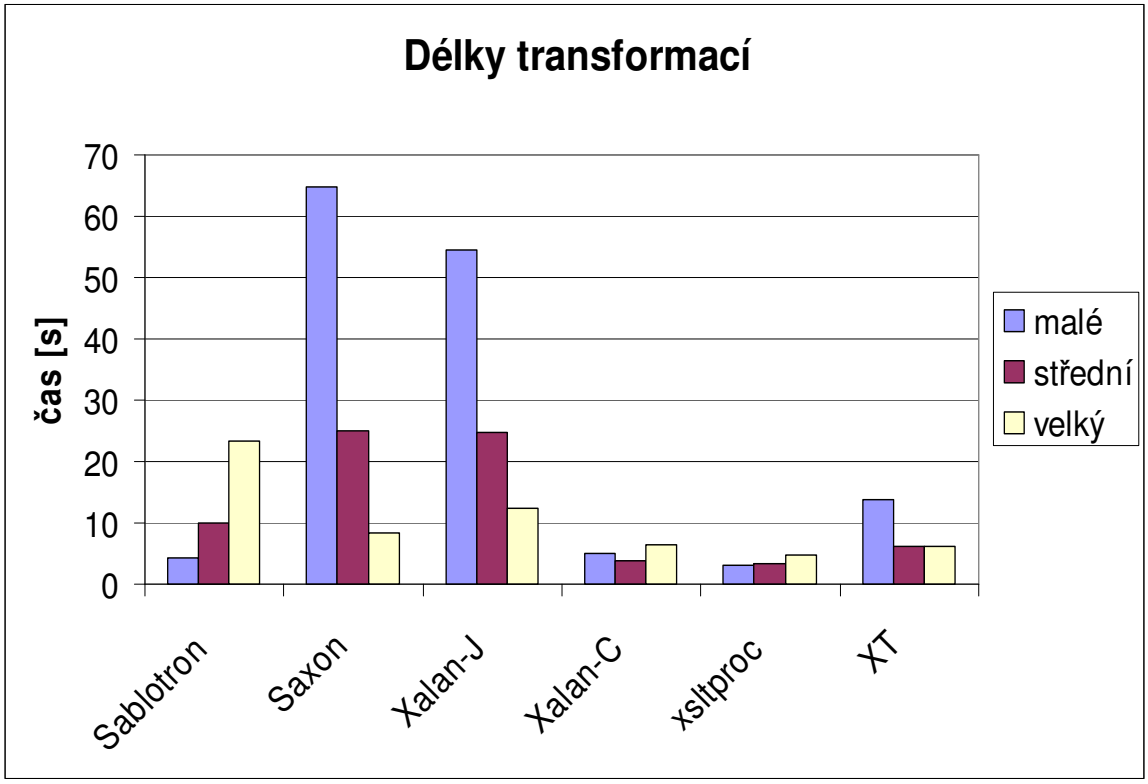
Při testování na velkém souboru dosáhl s časem pod pět sekund nejlepšího výsledku procesor xsltproc. Procesory Xalan-C a XT skončily s velmi podobnými časy, v okolí šesti sekund. Následoval Saxon a Xalan-J, který již překročil desetisekundovou hranici. Procesor Sablotron skončil transformaci až po dvaceti třech sekundách s tím, že jeho transformovaný dokument měl 20 MB, naproti tomu velikost souborů vytvořených ostatními procesory se pohybovala okolo 9 MB. Rozdílné velikosti HTML souborů byly zapříčiněny jejich odlišnou strukturou. Saxon a Sablotron tvoří HTML kód strukturovaně, tj. značky, které jsou uvnitř jiných, odsazují, zachovávají tedy tzv. bílé znaky. Sablotron ještě navíc za značky <u> a <br> vkládá nový řádek. Ostatní procesory – Xalan, Xalan-C, xsltproc a XT, generovaly kód nestrukturovaný. XT pak vkládal nový řádek za značku <br> a za počáteční tag prázdných párových značek. Xalan generoval stejný kód jako XT s rozdílem, že oba tagy prázdných párových značek nechával na jednom řádku. Procesor xsltproc generoval stejný kód jako Xalan, pouze za značkou <br> nebyl nový řádek. Výsledné HTML stránky všech procesorů byly v prohlížeči zobrazovány stejně.

Při zpracování dvaceti středně velkých souborů byl s časem 3.37 s opět nejrychlejší procesor xsltproc, avšak s velmi podobným časem skončil i Xalan-C, konkrétně 3,73 s. Procesor XT dosáhl téměř stejného času jako při zpracování velkého souboru, tj. zhruba šest sekund. Procesor Sablotron tentokrát již bez problému zvládl transformaci za bezmála deset sekund. Jako velmi pomalé se ukázaly Saxon a Xalan-J, které dosáhly času okolo dvaceti pěti sekund.

Nakonec procesory transformovaly sedmdesát malých souborů. Stejně jako v předchozích dvou případech, byl i tentokrát nejrychlejší xsltproc. Opět zvládl transformaci v čase okolo tří sekund. Jako druhý skončil Sablotron se čtyřmi sekundami. Následoval Xalan-C s pěti sekundami a XT se třinácti. Nejpomalejší se opět ukázaly Xalan-J s padesáti čtyřmi sekundami a Saxon, který jako jediný překročil jednu minutu s téměř šedesáti pěti sekundami.

Tab. 4 - Naměřené časy všech transformací

	Sablotron	Saxon	Xalan-J	Xalan-C	xsltproc	XT
Velký	23,433	8,271	12,495	6,487	4,758	6,277
Střední	9,917	25,038	24,667	3,734	3,37	6,288
Malé	4,306	64,701	54,617	4,945	2,984	13,724



Obr. 12 – Časy transformací při měření výkonu

8. Testování procesorů se složitými styly

8.1 Soubory pro testování

Pro testování procesorů se složitými styly jsem využil výše zmiňovaný DocBook. Konkrétně soubory docbook.xsl a chunk.xsl. Jako vstupní dokument jsem vytvořil zhruba 10 kB velký soubor kniha.xml, který obsahuje část textu této práce a je upraven docbookovými značkami.

Kniha.xml obsahuje kořenový uzel <book>, za ním následuje <bookinfo>, ve kterém jsou informace o dokumentu jako jeho název <title> a autor <autor>, který má v našem případě ještě další potomky <firstname> a <surname>.

Dalšími značkami jsou <preface> obsahující úvod, <chapter>, kde jsou jednotlivé kapitoly, jež se dají dále dělit na podkapitoly, v mém případě pomocí tagu <sect3>. Na konci dokumentu jsou přílohy ohraničené značkou <appendix>. Všechny zmiňované uzly v tomto odstavci mají ještě potomky <title>, kde je titulek a <char>, ve kterém je vlastní text.

```
<book lang="cs">
<bookinfo>
        <title>Testovací dokument</title>
        <author>
                <firstname>Rostislav</firstname>
                <surname>Rež</surname>
        </author>
</bookinfo>
<preface id="p01">
        <title>Úvod</title>
        <para>Odstavec textu.</para>
        <para>...</para>
</preface>
<chapter id="01">
        <title>Co je XSLT?</title>
        <para>…. </para>
</chapter>
```

Ukázka struktury souboru kniha.xml

Styl docbook.xsl vytvoří ze souboru kniha.xml jeden HTML dokument s odkazy na jednotlivé kapitoly a přílohy, to včetně nadpisů a číslování. Za zmínku stojí, že styl docbook.xsl při použití načítá ještě dalších 49 stylových dokumentů z DocBooku, které mají celkem velikost 1,1 MB.

Druhý používaný styl chunk.xsl vygeneroval obdobný HMTL dokument s tím rozdílem, že vytvoří úvodní stránku index.html, kde jsou pouze odkazy na jednotlivé kapitoly, přílohy a jiné části dokumentu, které jsou pak na samostatných HTML stránkách. Chunk.xsl do sebe importuje styly docbook.xsl a chunk-common.xsl. Načítá pak ještě soubor manifest.xsl.

8.2 Testování procesorů

Testování probíhalo obdobně jako v případě transformace velkého souboru (wurfl.xml) s tím rozdílem, že jsem použil jiné vstupní a výstupní soubory.

Sablotron

U Sablotronu došlo při všech pokusech k chybě a transformace neproběhla. Procesor vypsal následující chybu:

```
Error [code:46] [URI:///file/ ‘cesta ‘ /param.xsl] [line:4644] [node:element ‘<xsl:when>’] variable ‘stylesheet.result.type’ not found
```

Chybové hlášení procesoru Sablotron

Tato chyba se opakovala i při zkoušení se staršími verzemi procesoru a DocBooku. Na jejím základě byla na http://bugzilla.gingerall.cz/ založena nová chyba (ID 3215).

Saxon 6.5.3

Tato starší verze procesoru Saxon zvládla obě transformace bez problémů.

Saxon 8.8

Při testování se stylem docbook.xsl proběhlo vše bez problémů. Ty nastaly při zpracovávání stylu chunk.xsl. Při něm procesor selhal a vypsal chybovou hlášku.

```
Don’t know how to chunk with SAXON 8.8 from Saxonica
Processing terminated by xsl:message at line 41 in chunker.xsl
```

Chybové hlášení procesoru Saxon 8.8

Důvod, proč transformaci nezvládl je, že docbookvé styly jsou psané pro XSLT 1.0 procesory. Avšak Saxon 8.8 pracuje jako procesor odpovídající chystané specifikaci XSLT 2.0.

Xalan-C

Stejně jako Saxon 8.8 i Xalan-C zvládl bez problému transformaci se stylem docbook.xsl, ale při zpracování chunk.xsl skončil s následujícím hlášením.

```
XSLT Message: Don’t know how to chunk with Apache Software FundationSource tree node: preface (file:/// ‘cesta’ /chunker.xsl, line 63, column 36)
ElemMessageTerminateExpression: Don’t know how to chunk with Apache Software FundationSource tree node: preface (file:/// ‘cesta’ /chunker.xsl, line 63, column 36)
```

Chybové hlášení procesoru Xalan-C

Tento procesor selhal, protože při zpracování stylu chunk.xsl jsou potřeba rozšíření, která jsou implementována pouze v Java verzi procesoru.

Xalan-J

Tato Java verze procesoru Xalan zvládla bez problémů obě transformace.

Xsltproc

Stejně jako předchozí i tento procesor uspěl jak při zpracování stylu docbook.xsl, tak při chunk.xsl.

XT

Při testování obou stylů procesor skončil s chybovou hláškou.

```
File:/ ‘cesta’ /common.xsl:1205 no such function  key
```

Chybové hlášení procesoru XT

Stalo se tak, protože v procesoru XT není implementována funkce *key()*, kterou docbookové styly používají.

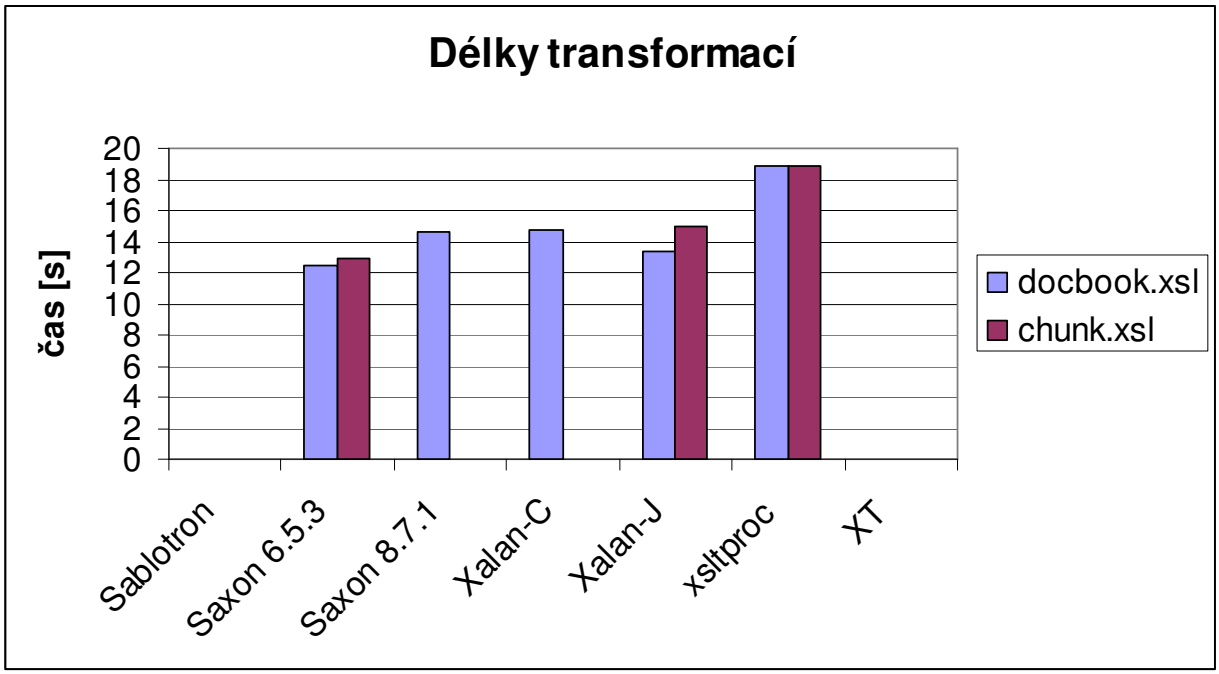
8.3 Výsledky měření a převodů

Tab. 5 – Naměřené časy všech převodů

t [s]	Sablotron	Saxon 6.5.3	Saxon 8.8	Xalan-C	Xalan-J	xsltproc	XT
docbook.xsl	----------	12,443	14,683	14,776	13,421	18,879	-----
chunk.xsl	----------	12,906	----------	---------	14,985	18,854	-----

Při transformování pomocí stylu docbook.xsl se s časem 12,44 s jako nejrychlejší jevil procesor Saxon 6.5.3. Přibližně o sekundu pomalejší byl Xalan-J. S časem okolo 14,7 s skončily Saxon 8.8 a Xalan-C. Nejdéle trvala transformace procesoru xsltproc, a to téměř 19s. Převod vůbec nezvládl Sablotron a stejně tak procesor XT.

Druhým složitým stylem byl chunk.xsl, ten zpracovaly bez problémů pouze tři testované procesory, a to Saxon 6.5.3 s téměř třinácti sekundami, který byl opět nejrychlejší. Xalan-J s patnácti sekundami. A opět nejpomalejší xsltproc, který transformoval prakticky za stejnou dobu jako při stylu docbook.xsl. Chunk.xsl tedy nezvládly zpracovat procesory Sablotron, Saxon 8.8, Xalan-C a XT.



Obr. 13 – Časy transformací při testování se složitými styly

9. Testování češtiny a výstupních formátů

Abych mohl u jednotlivých procesorů otestovat funkčnost a schopnost pracovat s dokumenty v českém jazyce, bylo třeba nejprve vypracovat dokumenty a stylové předpisy, které bych následně zpracovával. Pro mé potřeby jsem použil převod dokumentu XML do formátů HTML, TXT a úpravu původního XML dokumentu, konkrétně výběr určitých uzlů.

9.1 Dokument XML

Jako základ jsem vytvořil XML dokument prezentovaný jako sbírka CD, kde každé CD ve sbírce má osm dalších parametrů, přičemž dva z nich mají každý po jednom atributu. Tento dokument jsem vyhotovil ve dvou verzích kódování, windows-1250 a UTF-8. Stejně tak jsem vytvořil i dvě verze šablon pro úpravu XML a převod na ostatní formáty.

XML s kódováním windows-1250

Při použití kódování windows-1250 můžeme použít českou diakritiku (háčky a čárky).

```
<?xml version="1.0" encoding="windows-1250"?>
<!DOCTYPE SBIRKA SYSTEM "cd.dtd">
<SBIRKA>
<CD>
        <NAZEV>Černý kočky, mokrý žáby</NAZEV>
        <INTERPRET>Lucie</INTERPRET>
        <ROK>1998</ROK>
        <CENA MENA="Kc">200</CENA>
        <DELKA JEDNOTKA="minut">52</DELKA>
        <POCETPISNI>13</POCETPISNI>
        <ZANR>Rock</ZANR>
        <LABEL>Universal</LABEL>
</CD>
```

Ukázka souboru cd.xml s kódováním windows-1250

XML s kódováním UTF-8

Protože některé XSLT procesory nepodporují kódování windows-1250, vytvořil jsem dokument s kódováním UTF-8. Jelikož si procesory neporadí při použití tohoto kódování s dokumenty s českou diakritikou, upravil jsem dokument XML tak, že neobsahoval české znaky.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SBIRKA SYSTEM "cd.dtd">
<SBIRKA>
<CD>
        <NAZEV>Cerny kocky, mokry zaby</NAZEV>
        <INTERPRET>Lucie</INTERPRET>
        <ROK>1998</ROK>
        <CENA MENA="Kc">200</CENA>
        <DELKA JEDNOTKA="minut">52</DELKA>
        <POCETPISNI>13</POCETPISNI>
        <ZANR>Rock</ZANR>
        <LABEL>Universal</LABEL>
</CD>
```

Ukázka souboru cd.xml s kódováním UTF-8

9.2 Šablony pro převod

Jak již bylo uvedeno, vytvořil jsem pro testování tři druhy stylových dokumentů, každý ve dvou verzích. Pro kódování windows-1250 a pro UTF-8. Protože jsou obě verze dokumentů téměř totožné a jejich podstata je stejná, popíšu je najednou. Dále zde nebudu uvádět celé výpisy šablon, ty jsou na přiloženém CD.

Šablona pro převod na HTML

Jako všechny ostatní stylové dokumenty začíná i tento hlavičkou, ve které je informace o verzi XML, stylu, jmenný prostor a kódování.

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Hlavička stylového dokumentu

Následuje instrukce určující výstupní formát:

<xsl:output method="html"/>

Po ní pak kombinace instrukcí pro zpracování a HTML tagů. Z XML dokumentu je vytvořena tabulka obsahující všechny jeho prvky, a to pomocí HTML značek <table></table>,<tr> </tr>a <td></td> . Výsledná HTML stránka je ještě upravena několika dalšími HTML značkami a jednotlivá CD jsou seřazena abecedně dle jejich názvu, a to pomocí následujícího XSL výrazu:

<xsl:sort select="Nazev" lang="cs"/>

Díky atributu *lang* s parametrem „cs" by se měla při třídění dodržovat pravidla českého jazyka. To například znamená, že „ch" je bráno jako jedno písmeno a ne jako dvojice písmen „c" a „h".

Šablona pro převod na text

Tato šablona umožňuje převést XML dokument na prostý neformátovaný text. Atribut *method* instrukce *output* tedy obsahuje hodnotu „text".

<xsl:output method="text"/>

Dále se vybírají jednotlivé uzly, respektive jejich textový obsah. To je proloženo instrukcí <xsl:text>, pomocí které do výstupního dokumentu vkládáme doplňující text. Díky němu jsou ve výsledném textu souvislé věty.

Šablona pro úpravu XML

Tento styl se od ostatních liší především tím, že jeho cílem není vytvořit dokument v jiném formátu, ale pouze upravit vstupní XML dokument. Instrukce určující výstup měla tedy tento tvar:

<xsl:output method="xml"/>

Po ní následují instrukce vybírající, které uzly budou ve výstupním dokumentu.

9.3 Testování procesorů

Jako u předchozích i u těchto testování jsem vytvořil dávkové soubory, které jsem pak pomocí příkazu *sh* spouštěl v Cygwin. Měření času opět zajišťoval příkaz *time*. Naměřené hodnoty a informace, zda procesor zvládl transformaci, jsou v následujících tabulkách.

Tab. 6 – Výsledky procesoru ***Sablotron***

Stylový dokument	cdh.xsl		cdt.xsl		cdx.xsl	
Kódování	UTF-8	win-1250	UTF-8	win-1250	UTF-8	win-1250
Transformace	OK	OK	OK	OK	OK	OK
Délka [ms]	117	118	92	92	100	102

Tab. 7 – Výsledky procesoru ***Saxon***

Stylový dokument	cdh.xsl		cdt.xsl		cdx.xsl	
Kódování	UTF-8	win-1250	UTF-8	win-1250	UTF-8	win-1250
Transformace	OK	OK	OK	OK	OK	OK
Délka [ms]	1075	1090	911	911	965	919

Tab. 8 – Výsledky procesoru ***Xalan-J***

Stylový dokument	cdh.xsl		cdt.xsl		cdx.xsl	
Kódování	UTF-8	win-1250	UTF-8	win-1250	UTF-8	win-1250
Transformace	OK	OK	OK	OK	OK	OK
Délka [ms]	911	911	760	758	810	811

Tab. 9 – Výsledky procesoru ***Xalan-C***

Stylový dokument	cdh.xsl		cdt.xsl		cdx.xsl	
Kódování	UTF-8	win-1250	UTF-8	win-1250	UTF-8	win-1250
Transformace	OK	OK	OK	OK	OK	OK
Délka [ms]	124	118	93	93	100	101

Tab. 10 – Výsledky procesoru ***xsltproc***

Stylový dokument	cdh.xsl		cdt.xsl		cdx.xsl	
Kódování	UTF-8	win-1250	UTF-8	win-1250	UTF-8	win-1250
Transformace	OK	s chybami	OK	s chybami	OK	s chybami
Délka [ms]	103	109	82	83	85	87

Z tabulky výsledků testu je patrné, že procesor zvládl všechny transformace, avšak českou diakritiku v kódování windows-1250 nezobrazoval správně.

*Tab. 11 – Výsledky procesoru **XT***

Stylový dokument	cdh.xsl		cdt.xsl		cdx.xsl	
Kódování	UTF-8	win-1250	UTF-8	win-1250	UTF-8	win-1250
Transformace	OK	----------	OK	----------	OK	----------
Délka [ms]	302	----------	217	----------	218	----------

Tento procesor nezvládá kódování windows-1250, tedy ani znaky s háčky a čárkami. Při pokusu o transformaci takového dokumentu hlásí chybu: „unsupported encoding“

*Tab. 12 – Výsledky procesoru **XT s úpravou pro český jazyk***

Stylový dokument	cdh.xsl		cdt.xsl		cdx.xsl	
Kódování	UTF-8	win-1250	UTF-8	win-1250	UTF-8	win-1250
Transformace	OK	OK	OK	OK	OK	OK
Délka [ms]	708	709	607	605	608	605

Tato rozšířená verze si již poradila s oběma typy kódování.

9.4 Výsledky testování

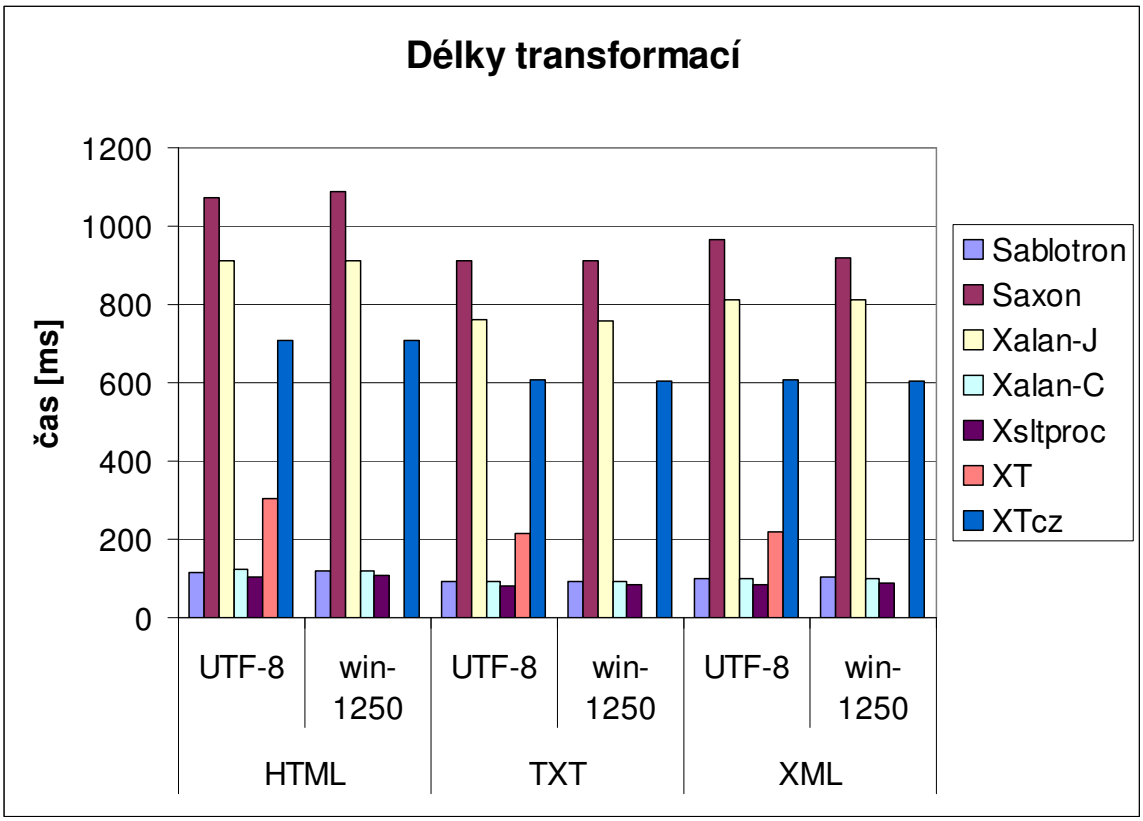
Časy transformací dokumentů, které se lišily pouze v použitém kódování, resp. v tom, zda v nich byla či nebyla použitá česká diakritika, byly prakticky stejné. Z pohledu výstupních formátů byla nejpomalejší transformace do HTML, o něco rychlejší byla úprava XML a nejrychleji procesory zvládaly převod do textového formátu.

Při tomto testování byl v průměru nejpomalejší procesor Saxon, jehož doba transformace do HTML jako jediná přesáhla jednu sekundu. Po něm následoval Xalan-J, XTcz, XT, který však neuměl transformovat dokumenty s kódováním windows-1250. O něco rychlejší pak byly Xalan-C, Sablotron a nejrychlejší xsltproc, který sice transformace zvládl, ale české znaky nezobrazoval správně.

Při ověřovaní řazení pomocí elementu *xsl*:*sort* s parametrem „cs“ jsem u jednotlivých procesorů zjistil následující: Sablotron, Saxon, Xalan-J, XT a XTcz řadily názvy dle českých pravidel, tj. „ch“ braly jako jedno písmeno a řadily jej mezi „h“ a „i“. Název začínající na „č“ pak správně vkládaly za názvy na „c“ (kromě procesoru XT, který jsem testoval na dokumentu bez českých znaků). Procesory Xalan-C a xsltproc řadily názvy začínající na „ch” mezi názvy na „c”, braly tedy „ch” jako dvojici znaků „c” a „h”. Název začínající na „č“ pak řadily až na úplný konec za „z“. Česká pravidla tedy nedodržovaly.

Tab. 13 – Tabulka časů všech transformací

	HTML		TXT		XML	
	UTF-8	win-1250	UTF-8	win-1250	UTF-8	win-1250
Sablotron	117	118	92	92	100	102
Saxon	1075	1090	911	911	965	919
Xalan-J	911	911	760	758	810	811
Xalan-C	124	118	93	93	100	101
xsltproc	103	109	82	83	85	87
XT	302	----------	217	----------	218	----------
XTcz	708	709	607	605	608	605



Obr. 14 – Časy transformací při testování se složitými styly

Závěr

Cílem této bakalářské práce bylo stručně popsat základy jazyka XSLT a jeho zpracování. Vyhledat volně šiřitelé XSLT procesory, otestovat je na dokumentech a stylových předpisech. Vyhodnotit dosažené výsledky a doporučit vhodný procesor pro zpracování českých textů.

Při testování výkonu jsem získal tyto výsledky. Jako nejrychlejší se jevil procesor xsltproc, a to jak při testovaní na velkém, středně velkých, tak i na malých souborech. Žádná jeho transformace nepřesáhla pět sekund. Druhý, v průměru jen o málo vteřin pomalejší, byl procesor Xalan-C, jehož časy převodů nepřesáhly deset sekund. Na třetím místě skončil procesor XT, který transformoval velký i středně velké soubory v prakticky stejném čase šesti sekund. Horších výsledků dosáhl při transformování malých souborů, které mu trvaly téměř čtrnáct sekund. Procesor Sablotron s časy pod deset sekund pro střední a malé soubory skončil na čtvrtém místě, avšak doba transformace velkého souboru přesáhla dvacet tři sekund. Kromě toho byl výsledný dokument o 11 MB větší než u všech ostatních. Výkonnostně nejhůře dopadly javové procesory Saxon a Xalan-J, které byly, až na výjimku zmiňovanou u Sablotronu, vždy nejpomalejší. Za jejich špatné výsledky mohl především pomalý start zapříčiněný nutností spouštět Javu při každém startu procesoru, což se projevilo nejvíce při testování na malých souborech, kde se časy obou procesorů pohybovaly okolo jedné minuty.

Další část testování se týkala složitých docbookových stylů, konkrétně šlo o docbook.xsl a chunk.xsl. S nimi si nejlépe poradila starší verze procesoru Saxon, Saxon 6.5.3 obě transformace zvládl, a to v nejlepším čase dvanácti sekund. O zhruba vteřinu pomalejší byl Xalan-J. Třetím a posledním procesorem, který zvládl zpracovat oba styly, byl xsltproc, a to za osmnáct sekund. Procesory Xalan-C a Saxon 8.8 zvládly pouze styl docbook.xsl, a to shodně v čase okolo čtrnácti sekund. Procesory Sablotron a XT neuspěly ani při jedné transformaci.

Jako poslední jsem testoval převody do HTML, na textový dokument a úpravu XML, a to jak při použití kódování UTF-8, tak i na dokumentech s kódováním windows-1250. Dokumenty bez české diakritiky zvládly bez problémů transformovat všechny procesory, a to v časech odpovídajících testování výkonu.

Při použití kódování windows-1250 neuspěly pouze procesory XT a xsltproc. XT transformaci vůbec nedokončil a xsltproc sice dokument transformoval, avšak ve výsledku byly české znaky špatně zobrazeny. Procesor XT je však dostupný v úpravě, která kódování windows-1250 podporuje.

Pro transformování velkých XML dokumentů s relativně jednoduchými styly bych doporučil použít procesory napsané v jazyce C, konkrétně xsltproc nebo Xalan-C, které nebrzdí používání Javy. Na zpracování složitých stylů je pak nejvhodnější Xalan-J nebo Saxon 6.5.3. Pro dokumenty v českém jazyce se jeví jako nejlepší použít Xalan-C, či Sablotron, které zvládají transformace v nejlepších časech.

Jak jsem již uvedl v úvodu, tato práce se týká oboru, které se velmi rychle vyvíjí. Příkladem toho byla i skutečnost, že jsem byl nucen v průběhu psaní práce několikrát předělávat praktickou část, protože vyšla nová verze jednoho z procesorů. Stejně tak bylo třeba do poslední chvíle upravovat i tento text.

Použité zdroje

[1] Holzner, Steven. *XSLT příručka internetového vývojáře*. Computer Press, Praha 2002, 1. vydání. ISBN 80-7226-600-4

[2] Grusová, Lucie. *XML pro úplné začátečníky*. Computer Press, Praha 2002, 1. vydání. ISBN-80-7226-697-7

[3] Kosek, Jiří. *XSLT v příkladech*. Dostupné z URL: http://www.kosek.cz/xml/xslt/index.html

[4] Kosek, Jiří. *Teorie a praxe značkovacích jazyků*. Dostupné z URL: http://www.kosek.cz/vyuka/izi238/slidy/xsl/frames.html

[5] Pavlovič, Jan. *Informace o XML technologiích na FI*. Dostupné z URL: http://www.fi.muni.cz/~xpavlov/xml/info.html

[6] Cimprich, Petr. *Váhání kolem XPath 2.0*. Dostupné z URL: http://www.root.cz/clanky/akta-x-0408/

[7] Cimprich, Petr. *XPath 2.0 a XSLT 2.0 se blíží*. Dostupné z URL: http://www.root.cz/clanky/xpath-20-a-xslt-20-se-blizi/

[8] Albrechtová, Zdeňka. *XSLT*. Dostupné z URL: http://www.gis.zcu.cz/studium/pok/XSLT/index.html

[9] Pichlík, Roman. *XSLT, svět transformací*. Dostupné z URL: http://www.sovavsiti.cz/2003/xslt-1.html

[10] Bříza, Petr. *Kompletní průvodce XSLT*. Dostupné z URL: http://interval.cz/serialy/kompletni-pruvodce-xslt/

[11] XLS Transformace (XSLT) Verze 1.0. Dostupné z URL: http://xslt.kulna.cz/

[12] Webové stránky procesoru Sablotron. Dostupné z URL: http://www.gingerall.org/sablotron.html

[13] Webové stránky procesoru xsltproc. Dostupné z URL: http://xmlsoft.org/

[14] Webové stránky procesoru XT. Dostupné z URL: http://www.blnz.com/xt/index.html

[15] Webové stránky procesoru Saxon. Dostupné z URL: http://saxon.sourceforge.net/

[16] Webové stránky procesoru Xalan. Dostupné z URL: http://xml.apache.org/xalan-j/

[17] Webové stránky W3C konsorcia. Dostupné z URL: http://www.w3.org/

Příloha - Obsah přiloženého CD

Na CD jsou dva základní adresáře, *Práce a Testy.* V první složce se nachází elektronická verze tohoto dokumentu ve formátech DOC a PDF. Druhý adresář je rozdělen na další složky, které odpovídají jednotlivým testům procesorů, jsou jimi tedy C*estina, DocBook* a V*ykon*.

Složka C*estina* obsahuje jednak dávkové soubory potřebné pro spouštění jednotlivých transformací, dále pak podadresáře

- Out – do tohoto adresáře se ukládají výstupní dokumenty, vzniklé jednotlivými transformacemi
- Procesory – zde jsou uloženy jednotlivé procesory v podadresářích Sablotron, Saxon, Xalan-C, Xalan-J, Xsltproc, XT a XTcz
- Xml – v této složce jsou uloženy dokumenty cd.xml a cd.dtd, které se zpracovávají při transformaci. Současně jsou zde další adresáře UTF-8, obsahující verze dokumentů bez diakritiky a win-1250, kde jsou dokumenty s diakritikou.
- Xsl – tato složka obsahuje soubory se stylovými předpisy, těmi jsou

  cdh.xsl – styl pro transformaci do HTML

  cdt.xsl – styl pro transformaci do textového souboru

  cdx.xsl – styl pro úpravu na nový XML dokument

  Dále tento adresář obsahuje složky UTF-8 a win-1250, opět s dvěma verzemi stylových předpisů.

Adresář *DocBook* obsahuje dávkové soubory pro spouštění transformací, dále pak podadresáře

- DocBook – v této složce je uložen DocBook používaný při testování
- Out – do tohoto adresáře se ukládají výstupní dokumenty, vzniklé jednotlivými transformacemi.
- Procesory – zde jsou uloženy jednotlivé procesory v podadresářích Sablotron, Saxon 6.5.3, Saxon 8 Xalan-C, Xalan-J, Xsltproc a XT.

- Xml – v této složce je uložen dokument kniha.xml, na který se aplikují docbookové styly

Adresář V*ykon* obsahuje, stejně jako předešlé, dávkové soubory a další podadresáře, těmi jsou

- Male – v této složce je uloženo sedmdesát malých XML souborů.
- Out – do tohoto adresáře se ukládají výstupní dokumenty vzniklé jednotlivými transformacemi.
- Procesory – zde jsou uloženy jednotlivé procesory v podadresářích Sablotron, Saxon, Xalan-C, Xalan-J, Xsltproc a XT.
- Stredni – v této složce je uloženo dvacet středně velkých XML souborů
- Wurfl – v tomto adresáři je uložen soubor wurfl.xml, jeho DTD a styl convert_2_hml.xsl, který jsem používal při testování.

V adresářích *out* jsou uloženy výsledky mnou provedených transformací.